References

# PGSLib: A Library for Portable, Parallel, Unstructured Mesh Simulations

Robert C. Ferrell [†] Douglas B. Kothe [‡] John A. Turner [§]

**Abstract**

We have developed a programming model and a library of parallel routines (PGSLib) to support parallel unstructured mesh simulations. The model and library was developed for DOE's ASCI project. Current applications using PGSLib include: JTpack90 (a parallel linear algebra package), TELLURIDE (metal casting simulation) and CHAD (high-speed reacting flow simulation). This paper will describe our implementation strategy and show some performance results.

## 1 Introduction

Finite element simulations use irregular data structures with data access patterns not known until run time. When executing on multiple processors different processors will read and write the same memory location. On distributed memory systems software over and above the CPU load and store instructions must be used to read and write the memory of another processor. On systems with shared memory hardware loads and stores may be adequate, but a (possibly system dependent) memory consistency model must be chosen and programmed correctly.

These tasks burden the programmer with issues not directly germane to developing a finite element simulation. Fortunately finite element simulations require only a relatively small amount of general indirect addressing functionality. (Typically indirect addressing will involve interprocessor communication.) It is possible, and desirable, to isolate the required functionality and provide it in a library. That is the approach of PGSLib (Parallel Gather & Scatter Library), described here.

## 2 Portability and Performance

General interprocessor memory accesses cannot be implemented by accepted, standard, compilers such as FORTRAN90. On distributed memory machines it is possible to use HPF (High Performance Fortran) for most operations. However, HPF performance is generally not adequate and HPF is not ubiquitous. Many shared memory systems have parallelizing compilers which can provide most of the required functionality. However, the semantics for directing parallel operations are not standardized. In addition, performance

[*] Supported by the Department of Energy (DOE) Accelerated Strategic Computing Initiative (ASCI) Program

[†] Cambridge Power Computing Associates, Ltd., 2 Still St., Brookline, MA 02146, *ferrell@cpca.com*

[‡] Los Alamos National Laboratory, Fluid Dynamics Group T-3, MS B216, Los Alamos, NM 87545, *dbk@lanl.gov*

[§] Los Alamos National Laboratory, Transport Methods Group X-TM, MS B226, Los Alamos, NM 87545, *turner@lanl.gov* and optimizations vary from system to system. Thus, an implementation which does execute correctly on different systems may have widely varying performance characteristics.

By isolating the pieces of code which are system dependent it is possible to provide portable code more easily. Furthermore, good parallel algorithms and system specific optimizations can more easily be implemented in just a few places. Since similar functionality is required of many different applications it makes sense to provide an application independent library to provide the required functionality.

## 2.1 Parallel Gather/Scatter Library

PGSLib is a library to support portable, parallel finite element simulations. It includes functionality which is both 1) required of unstructured mesh simulations, and 2) relevant to parallel computation.

At its core, the purpose of PGSLib is to support the gather and scatter operations required by most finite element simulations. These are, for a gather:

```
DO I = 1,Nelements
     DO J = 1, NodesPerElement
          Ele_Dest(J,I) = Node_Source(Neighbor(J,I))
     ENDDO
ENDDO
```

and for a scatter:

```
DO I = 1,Nelements
     DO J = 1, NodesPerElement
          Node_Dest(Neighbor(J,I)) = Node_Dest(Neighbor(J,I)) +
                                     Ele_Source(J,I)
     ENDDO
ENDDO
```

In these FORTRAN fragments `Node_Source` and `Node_Dest` are one dimensional arrays. `Ele_Dest` and `Ele_Source` are two dimensional arrays. `Neighbor` is a two dimensional integer array of the same shape as `Ele_Dest` and `Ele_Source`. Since the arrays are distributed to all available memory `Neighbor` may point to any node, so these loops will, generically, represent memory accesses between processors. According to our philosophy of isolating code which does global addressing, we replace these loops by library calls. For the gather, we use

```
call PGSLib_Gather(Ele_Dest, Node_Source, Neighbor, <...>)
```

and for the scatter

```
call PGSLib_Scatter_SUM(Node_Dest, Ele_Source, Neighbor,<...>)
```

where `<...>` indicates other variables which we will not describe. The scatter operations may be SUM, MIN, MAX, OR, AND.

The PGSLib routines are object based, so the same routine name is used for REAL, REAL*8, INTEGER, LOGICAL data types, and for 1-D, 2-D and 3-D arrays.

## 3 Implementation

### 3.1 Object Based Programming with FORTRAN90

The current interface to PGSLib is in FORTRAN90, which has proven to be well suited to scientific programming. PGSLib is provided through the FORTRAN90 module PGSlib_Module. This provides strong type checking for all subroutine arguments. It also provides generic routine names so that developers need not be burdened with remembering type specific and dimension specific routine names. In addition, FORTRAN90 allows PGSLib routines to determine array sizes for array arguments, further removing a common source of errors. Finally, FORTRAN90 provides data hiding mechanisms.

### 3.2 Communication Primitives

The implementation uses a "send-to-queue" construct. This provides a useful building block for the gather and scatter routines as well as other routines provided by PGSLib, such as the parallel ranking routine, `PGSLib_GRADE_UP`.

In a typical operation each processor moves data into a (local) buffer, then data from the buffers are sent to their destinations, and finally the newly arrived data is transferred to its ultimate destination (through local operations). The source buffer for a gather operation is the destination buffer for a scatter operation, and vice versa. For either a gather or a scatter only a single pass through the finite element mesh is required.

The communication buffers may be considered ghost cells. The buffer transfer mechanism is exposed so that developers wishing to implement their own ghost cells can take advantage of PGSLib.

### 3.3 Optimizations

PGSLib has been optimized for distributed memory systems. For the gathers and scatters PGSLib uses an inspector/executor[1] model. Before using any particular global index a call to `PGSLib_GS_Setup` must be made. This routine:

1. Translates a global index into (processor, local index) pairs.
2. Aggregates messages so that any pair of processors exchange at most one message. This reduces the impact of message passing latency.
3. Schedules the communication, determining which data items a processor will send and which it will receive.

This state is stored in a *trace*. Only a single trace is required for any particular index. The same trace is used for both gather and scatter, and for all data types.

### 3.4 Data Locality

In any multiprocessor system it is imperative (for performance) that most memory accesses do not cause contention. On a distributed memory and cache based systems system it is important that most accesses are local. Typically, a finite element mesh is partitioned into submeshes. A submesh is assigned to the memory closest to the processor which does (most of) the computing on it. Usually the partitioning is done so as to minimize the boundary between submeshes.

PGSLib exploits whatever partitioning has been done. All routines work correctly on unpartitioned or randomly distributed meshes. However, if a mesh has been partitioned PGSLib exploits that.

### 3.5 Memory Usage

The philosophy of PGSLib is that, to the extent possible, memory management should be done by the host program. To that end, PGSLib accepts standard FORTRAN90 arrays as arguments. PGSLib specific arrays are not required. The buffers that PGSLib requires may be allocated by the user or by PGSLib, depending on parameters. The size of the buffers is proportional to the size of the boundary (surface of the submeshes). Consequently, for a high quality partitioning of a mesh the buffers are small (since one measure of quality includes minimizing the size of the boundary).

### 3.6 Transport Layer

PGSLib uses MPI (Message Passing Interface) to transport data between processors. The required MPI functionality has been deliberately minimized. Future computers may supply other communication mechanisms (*e.g.* distributed shared memory hardware). PGSLib is designed to adapt to other communication mechanisms readily.

### 3.7 Serial Simulator

Parallel application development is challenging both because the algorithms may be new and because development tools are less mature than those available for serial program development. To aid developers PGSLib provides a serial simulator. The simulator has the same interface as the parallel library. However, it runs on only on a single processor. Applications linked with the serial simulator can be debugged using any debugger used for serial programs.

The behavior of the serial simulator is defined as follows. PGSLib (parallel version) has defined outputs for any number of processors. The serial simulator gives the same results as parallel PGSLib on one processor. (This is true only for programs calling PGSLib correctly. No such guarantee is available for programs which are not using PGSLib correctly.)

The serial simulator makes it easier to develop new physics modules, for instance. Also, it allows developers to gradually incorporate the use of PGSLib. Once the transition is complete it is only necessary to re-link, not re-compile, the program to switch from the serial simulator to parallel PGSLib

## 4 Other Functionality in PGSLib

### 4.1 Reductions

The gather and scatter operations are needed for computing differential operators as well as matrix-vector multiplies. Many finite element simulations use iterative solvers as well (JTPack90)[2] Those require global reduction operations such as dot product. We've included a variety of reductions in PGSLib, with names and functionality motivated by FORTRAN90. The distinction is that these operate on distributed arrays. The supported reduction operations are:

| | |
|---|---|
| `PGSLib_DOT_PRODUCT` | |
| `PGSLib_MINVAL` | `PGSLib_MAXVAL` |
| `PGSLib_MAXLOC` | `PGSLib_MINLOC` |

### 4.2 Support for Mesh Distribution and Re-Ordering

Many mesh partitioners take node/node or element/element connectivity as input. Many mesh generation packages supply only element/node connectivity. The other connectivity must be constructed. Typical mesh partitioners return a partition number for each node

TABLE 1
*Implicit heat flow on 46,386-cell chalice mesh.* [a]

| Processors | CPU Time ($\mu$s/cell/cycle) | Speed Up | Efficiency |
|---|---|---|---|
| 1 | 5013 | 1.0 | 1.00 |
| 2 | 2169 | 2.3 | 1.15 |
| 4 | 1237 | 4.1 | 1.01 |
| 8 | 721 | 7.0 | 0.87 |

[a] 300 HHz Digital AlphaServer 8400

or each element of the mesh. A permutation vector must be determined and applied to the mesh so that it is organized into partitions. These operations, and other similar graph operations, often are done serially because they are done only at the start of a job, and hence execution time is not critical. However, a large mesh may be too large for a single serial process. PGSLib provides routines that can perform the necessary operations in parallel. Many of the routines were motivated by HPF_Library routines. The provided routines include:

1. `PGSLib_GRADE_UP` Performs parallel ranking of a distributed array. Also accepts a segment (in the HPF sense) and performs segmented parallel ranking.

2. `PGSLib_SUM_PREFIX`, `PGSLib_SUM_SUFFIX`, `PGSlib_PARITY_PREFIX` and `PGSLib_PARITY_SUFFIX` Perform scan operations on distributed arrays.

3. `PGSLib_CSHIFT`, `PGSLib_EOSHIFT` Global versions of the FORTRAN90 routines (for a restricted class of arrays).

4. `PGSLib_PERMUTE` A useful routine which permutes a distributed vector according to a distributed permutation vector.

## 5 Applications Using PGSLib

Telluride[3] is designed to simulate metal casting. The core physics includes the (low-speed) flow of molten material and the subsequent solidification. Telluride is described elsewhere in these proceedings. CHAD is designed for simulation of combustion engines. The core physics includes high speed reacting flow on a moving mesh. In addition to these, other applications in the ASCI program are exploring the use of or planning to use PGSLib.

## 6 Performance Results

We present three tests from the Telluride program. In these tests heat flow is solved using an implicit solver. JTPack provides the conjugate gradient routine. PGSLib provides the indirect addressing required for matrix-vector multiply as well as that required for difference operators in Telluride.

In each of the tables *Speed Up* is defined as $T_1/T_{NP}$ where $T_n$ is the time on $n$ processors and $NP$ is the number of processors. *Efficiency* is *Speed Up*/$NP$.

## References

TABLE 2

*Implicit heat flow on 16 × 16 × 192 mesh using a shared memory multiprocessor.* [a]

| Processors | CPU Time ($\mu$s/cell/cycle) | Speed Up | Efficiency |
|---|---|---|---|
| 1 | 583 | 1.0 | 1.00 |
| 2 | 258 | 2.3 | 1.13 |
| 3 | 162 | 3.6 | 1.20 |
| 4 | 129 | 4.5 | 1.13 |
| 6 | 93 | 6.3 | 1.04 |
| 8 | 69 | 8.4 | 1.06 |

[a] 300 MHz Digital AlphaServer 8400

TABLE 3

*Implicit heat flow on 16 × 16 × 320 mesh using a distributed memory multiprocessor.* [a]

| Processors | CPU Time ($\mu$s/cell/cycle) | Speed Up | Efficiency |
|---|---|---|---|
| 1 | 1113 | 1.0 | 1.00 |
| 2 | 635 | 1.8 | 0.88 |
| 10 | 124 | 9.0 | 0.90 |
| 20 | 65 | 17. | 0.86 |

[a] 67 MHz IBM SP2